Лекция 7. Указатели

Казахский национальный университет имени аль-Фараби

Кафедра компьютерных наук

Объектно-ориентированное программирование

Лектор: старший преподаватель, PhD, Кенжебек Ержан Ғалымжанұлы

Цель лекции

Цель лекции

Изучить понятие указателя, операции взятия адреса и разыменования, а также основные приёмы безопасной работы с указателями.

Ожидаемые результаты

1. Объявлять указатели и присваивать им адреса переменных
2. Применять операции & и *
3. Работать с несколькими указателями и нулевыми указателями
4. Отличать указатель от ссылки

План лекции

- Что такое указатели
- Простой пример указателей в C++
- Пример
- Операция разыменования
- Создание нескольких указателей
- Операция присваивание
- Нулевые указатели

Что такое указатели

Указатели представляют собой объекты, значением которых служат адреса других объектов (переменных, констант, указателей) или функций. Как и ссылки, указатели применяются для косвенного доступа к объекту. Однако в отличие от ссылок указатели обладают большими возможностями.

Для определения указателя надо указать тип объекта, на который указывает указатель, и символ звездочки *. Например, определим указатель на объект типа int:

```
int *p;
```

Пока указатель не ссылается ни на какой объект. При этом в отличие от ссылки указатель необязательно инициализировать каким-либо значением. Теперь присвоим указателю адрес переменной:

```
int x = 10;     // определяем переменную
int *p;         // определяем указатель
p = &x;         // указатель получает адрес переменной
```

Для получения адреса переменной применяется операция **&**. Что важно, переменная x имеет тип int, и указатель, который указывает на ее адрес, тоже имеет тип int. То есть должно быть соответствие по типу.

Простой пример

```cpp
#include <iostream>

int main()
{
    int x = 10;     // определяем переменную
    int *p;         // определяем указатель
    p = &x;         // указатель получает адрес переменной
    std::cout << "p = " << p << std::endl;
    return 0;
}
```

Консольный вывод программы:

```
p = 0x60fe98
```

В каждом отдельном случае адрес может отличаться, но к примеру, в моем случае машинный адрес переменной x - 0x60fe98. То есть в памяти компьютера есть адрес 0x60fe98, по которому располагается переменная x. Так как переменная x представляет тип **int**, то на большинстве архитектур она будет занимать следующие 4 байта (на конкретных архитектурах размер памяти для типа int может отличаться). Таким образом, переменная типа int последовательно займет ячейки памяти с адресами 0x60FE98, 0x60FE99, 0x60FE9A, 0x60FE9B.

Пример

И указатель **p** будет ссылаться на адрес, по которому располагается переменная x, то есть на адрес 0x60FE98.

Но так как указатель хранит адрес, то мы можем по этому адресу получить хранящееся там значение, то есть значение переменной x. Для этого применяется операция * или операция разыменования, то есть та операция, которая применяется при определении указателя. Результатом этой операции всегда является объект, на который указывает указатель. Применим данную операцию и получим значение переменной x:

```
#include <iostream>

int main()
{
    int x = 10;
    int *p;
    p = &x;
    std::cout << "Address = " << p << std::endl;
    std::cout << "Value = " << *p << std::endl;
    return 0;
}
```

Консольный вывод:

```
Address = 0x60fe98
Value = 10
```

Операция разыменования

Значение, которое получено в результате операции разыменования, можно присвоить другой переменной:

```
int x = 10;
int *p  = &x;
int y = *p;
std::cout << "Value = " << y << std::endl;    // 10
```

И также используя указатель, мы можем менять значение по адресу, который хранится в указателе:

```
int x = 10;
int *p = &x;
*p = 45;
std::cout << "x = " << x << std::endl;     // 45
```

Так как по адресу, на который указывает указатель, располагается переменная x, то соответственно ее значение изменится.

Создание нескольких указателей

```cpp
#include <iostream>

int main()
{
    short c = 12;
    int d = 10;
    short s = 2;

    short *pc = &c;          // получаем адрес переменной c типа short
    int *pd = &d;           // получаем адрес переменной d типа int
    short *ps = &s;         // получаем адрес переменной s типа short

    std::cout << "Variable c: address=" << pc << "\t value=" << *pc << std::endl;
    std::cout << "Variable d: address=" << pd << "\t value=" << *pd << std::endl;
    std::cout << "Variable s: address=" << ps << "\t value=" << *ps << std::endl;

    return 0;
}
```

В моем случае я получу следующий консольный вывод:

```
Variable c: address=0x60fe92 value=12
Variable d: address=0x60fe8c value=10
Variable s: address=0x60fe8a value=2
```

Операция: присваивание

Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя.

Присвоение указателю адреса уже рассматривалось в прошлой теме. Для получения адреса объекта используется операция **&**:

```
int a = 10;
int *pa = &a;   // указатель pa хранит адрес переменной a
```

При этом указатель и переменная должны иметь один и тот же тип, в данном случае это тип int.

Присвоение указателю другого указателя:

```
#include <iostream>
using std::cout;
using std::endl;

int main()
{
    int a = 10;
    int b = 2;

    int *pa = &a;
    int *pb = &b;

    cout << "Variable a: address=" << pa << "\t value=" << *pa << endl;
    cout << "Variable b: address=" << pb << "\t value=" << *pb << endl;

    pa = pb;    // теперь указатель pa хранит адрес переменной b
    cout << "Variable b: address=" << pa << "\t value=" << *pa << endl;

    return 0;
}
```

Когда указателю присваивается другой указатель, то фактически первый указатель начинает также указывать на тот же адрес, на который указывает второй указатель.

Нулевые указатели

Нулевые указатели

Нулевой указатель (null pointer) - это указатель, который не указывает ни на какой объект. Если мы не хотим, чтобы указатель указывал на какой-то конкретный адрес, то можно присвоить ему условное нулевое значение. Для создания нулевого указателя можно применять различные способы:

```
int *p1 = nullptr;
int *p2 = NULL;
int *p3 = 0;
```

Адрес указателя

Адрес указателя

Указатель хранит адрес переменной, и по этому адресу мы можем получить значение этой переменной. Но кроме того, указатель, как и любая переменная, сам имеет адрес, по которому он располагается в памяти. Этот адрес можно получить также через операцию **&**:

```
1  int a = 10;
2  int *pa = &a;
3  std::cout << "address of pointer=" << &pa << std::endl;         // адрес указателя
4  std::cout << "address stored in pointer=" << pa << std::endl;  // адрес, который хранится в указателе - адрес переменной a
5  std::cout << "value on pointer=" << *pa << std::endl;          // значение по адресу в указателе - значение переменной a
```

Указатели и ссылки

Указатель

- Хранит адрес переменной
- Может быть нулевым (nullptr)
- Можно переназначить на другой объект
- Доступ к значению — через операцию *

```
int x = 5;
int* p = &x;
*p = 7;   // x == 7
```

Ссылка

- Псевдоним существующей переменной
- Должна быть инициализирована при объявлении
- Нельзя переназначить на другой объект
- Используется как обычная переменная

```
int x = 5;
int& r = x;
r = 7;    // x == 7
```

Выводы

1. Указатель — переменная, хранящая адрес другой переменной в памяти.
2. Операция & возвращает адрес переменной, операция разыменования * даёт доступ к значению по адресу.
3. Несколько указателей могут ссылаться на одну переменную; присваивание копирует адрес, а не значение.
4. Нулевой указатель (nullptr) не указывает на объект; его нельзя разыменовывать, перед использованием указатель нужно проверять.
5. Указатель имеет собственный адрес в памяти и в отличие от ссылки может быть переназначен.

Контрольные вопросы

1. Что такое указатель и что он хранит?
2. Для чего служат операции & и *?
3. Как объявить указатель на int и записать в него адрес переменной?
4. Что произойдёт при присваивании одного указателя другому?
5. Что такое нулевой указатель и почему его нельзя разыменовывать?
6. Чем указатель отличается от ссылки?

Список литературы

Основная литература

1. Лафоре Р. Объектно-ориентированное программирование в C++. 4-е изд. — СПб.: Питер, 2004.
2. Страуструп Б. Язык программирования C++. Специальное издание. — М.: Бином, 2011.
3. Липпман С., Лажойе Ж., Му Б. Язык программирования C++. Базовый курс. 5-е изд. — М.: Вильямс, 2014.

Дополнительная литература

4. Шилдт Г. Полный справочник по C++. 4-е изд. — М.: Вильямс, 2006.
5. Страуструп Б. Программирование: принципы и практика с использованием C++. 2-е изд. — М.: Вильямс, 2016.
6. Майерс С. Эффективный и современный C++. — М.: Вильямс, 2016.

Электронные ресурсы

7. Учебник по C++ (learncpp): www.learncpp.com
8. Справочник по C++ и стандартной библиотеке: en.cppreference.com

- Спасибо за внимание!